1. **(Designing a CFG)**

   Let $L = \{x \in \{0,1\}^* : x \neq ww \text{ for any } w \in \{0,1\}^*\}$. First, any string of odd length is clearly not in $L$. Consider, on the other hand, a string $s$ of even length $2n$. Let $s[i]$ denote the $i^{th}$ symbol of $s$, so that $s \in L$ if and only if $s[i] \neq s[n+i]$ for some $1 \leq i \leq n$.

   The key observation is that for any such $i$, $s$ can be split into two odd-length substrings $s_1$ and $s_2$ (of lengths $2i-1$ and $n-2i+1$, respectively) such that $s[i]$ and $s[n+i]$ are their respective middle symbols. Conversely, if we take any two odd-length strings $s_1$ and $s_2$ and concatenate them to form a string $s = s_1 s_2$ of length $2n$, then the middle indices of $s_1$ and $s_2$ correspond to $s[i]$ and $s[n+i]$, respectively, for some $1 \leq i \leq n$ (namely, $i = (|s_1|+1)/2$).[1]

   We can then say that a string is in $L$ if and only if it is the concatenation of (equivalently, it can be split into) two odd-length strings whose middle symbols differ. The following grammar captures this formulation.

$$\begin{aligned} S &\rightarrow S_{odd} | S_{01} | S_{10} \\ S_{odd} &\rightarrow S_{odd} AA | A \\ S_{01} &\rightarrow M_0 M_1 \\ S_{10} &\rightarrow M_1 M_0 \\ M_0 &\rightarrow AM_0A | 0 \\ M_1 &\rightarrow AM_1A | 1 \\ A &\rightarrow 0 | 1 \end{aligned}$$

   Here, $S_{odd}$ generates all odd-length strings, $S_{01}$ generates $\{s_1 s_2 \in \{0,1\}^* : s_1, s_2$ have middle symbols 0, 1, resp.$\}$, $S_{10}$ generates $\{s_1 s_2 \in \{0,1\}^* : s_1, s_2$ have middle symbols 1, 0, resp.$\}$, and $M_0$ and $M_1$ generate odd-length strings with middle symbols 0 and 1, respectively.

2. **(Decoding a CFG)**

   2.1. Let $G$ denote the given grammar. Then $\mathcal{L}(G) = \{x \in \{0,1\}^* : N_0(x) = 2N_1(x)\}$.

   2.2. *Proof.* For concision, let us say a string $x$ satisfies $\mathcal{C}$ if $N_0(x) = 2N_1(x)$. The easy direction to show is $\mathcal{L}(G) \subset \{x \in \{0,1\}^* : w \text{ satisfies } \mathcal{C}\}$. Let $x \in \mathcal{L}(G)$. Take any $x \in \mathcal{L}(G)$. We show that $x$ satisfies $\mathcal{C}$ by inducing on the length of the shortest derivation of $x$ in $G$. If this length is 0, then $x = \varepsilon$ and we are done since $\varepsilon$ satisfies $\mathcal{C}$. If it is $\geq 1$, then consider the first step of a shortest derivation $S \stackrel{*}{\Rightarrow} x$. Suppose it is $S \Rightarrow 1S00$. Then by induction, the string generated by the $S$ on the RHS satisfies $\mathcal{C}$, so $x$ must also satisfy $\mathcal{C}$. A similar argument applies to all the other rules as well, so our induction is complete.

   Now let us prove that $\{x \in \{0,1\}^* : x \text{ satisfies } \mathcal{C}\} \subset \mathcal{L}(G)$. We do this using strong induction on the length of the string. For our base case, the shortest string satisfying $\mathcal{C}$ is $\varepsilon$, which we can generate by $S \Rightarrow \varepsilon$, so $\varepsilon \in \mathcal{L}(G)$. For our inductive case, take a string $x$ satisfying $\mathcal{C}$ such that $|x| \geq 3$.

   Suppose $x$ is of the form $1y00$ for some $y \in \{0,1\}^*$. Then $y$ must certainly satisfy $\mathcal{C}$, so by inductive hypothesis we have a derivation $S \stackrel{*}{\Rightarrow} y$. Adding a step to this gives us the derivation $S \Rightarrow 1S00 \stackrel{*}{\Rightarrow} 1y00 = x$, so $x \in \mathcal{L}(G)$ and we are done. Likewise, if $x = 00y1$ for some string $y$.

   Now, let $w[i \ldots j]$ denote the substring of $w$ from index $i$ to index $j$, inclusive. We can define the function $f_w : \{0, \ldots, |w|\} \rightarrow \mathbb{Z}$ by

$$f_w(i) = N_0(w[1 \ldots i]) - 2N_1(w[1 \ldots i])$$

[1] *Note:* You should try a few concrete examples to convince yourself of this fact.

so that $w$ satisfies $\mathcal{C}$ if and only if $f_w(0) = f_w(|w|)$. To visualize the graph of $f_w$, imagine walking along $w$ from left to right, starting at the origin. Each time you see a 0, move up one level, and each time you see a 1, move down two levels.

Suppose, then, that $x$ is of neither of the two forms above. Then it can take any of the following forms:

(1) $1 \dots 1$
(2) $1 \dots 10$
(3) $0 \dots 0$
(4) $01 \dots 1$

Consider case (1). We know that $f_x(0) = f_x(n) = 0$, so since $x[1] = x[n] = 1$, it follows that $f_x(1) = -2$ and $f_x(n-1) = 2$. But $f_x$ can only increase by 1 at a time, so it must hit 0 (i.e., the horizontal axis) at some point, that is, $f_x(i) = 0$ for some $2 \leq i \leq n-2$. But this implies that the substrings $x[1 \dots i]$ and $x[i+1 \dots n]$ both satisfy $\mathcal{C}$. By inductive hypothesis, they can then be derived in the grammar $G$. Combining these derivations with the rule $S \Rightarrow SS$ gives us a derivation of $x$.

Cases (2) and (4) also share the property that $f_x(1) < 0 < f_x(n-1)$, so the same reasoning applies to give us a derivation of $x$. Case (3), however, is a little different. We have $f_x(1) = 1 > 0 > -1 = f_x(n-1)$, so as before we are guaranteed that $f_x$ must move from positive to negative values at some point. But because $f_x$ drops 2 at a time, it might not *hit* 0, but instead jump directly from 1 to -1. If it *does* hit 0, then as before we are done. Otherwise, let $j$ be the index of the 1 that causes it to cross over, that is, $f_x(j-1) = 1$ and $f_x(j) = -1$. Then $f_x(1) = f_x(j-1)$, so $x[2 \dots j-1]$ satisfies $\mathcal{C}$. Likewise, $f_x(j) = f_x(n-1)$, so $x[j+1 \dots n-1]$ satisfies $\mathcal{C}$. By induction, they can be derived in $G$, so combining these derivations with the rule $S \Rightarrow 0S1S0$ gives us a derivation of $x$. This completes our induction and proof. □

3. **(CFL or not?)**

3.1. $L = \{a^n b^n c^m : n \leq m \leq 2n\}$ is not a CFL.

*Proof.* Suppose, to get a contradiction, that $L$ is context-free. Let $s = a^p b^p c^p$, where $p$ is the pumping length of $L$. Let $s = uvxyz$ be the division of $s$ specified by the Pumping Lemma. Since $|vxy| \leq p$, $vy$ cannot contain both $a$'s and $c$'s. If $vy$ contains neither, that is, it consists entirely of $b$'s, then pumping down will result in fewer $b$'s than $a$'s. If $vy$ contains $a$'s (and possibly $b$'s, but no $c$'s), then pumping up will break the condition $n \leq m$. Finally, if $vy$ contains $c$'s, then pumping down will break the same condition. Hence, we have a contradiction of the Lemma, so $L$ cannot be a CFL. □

3.2. $L = \{a^n b^m : n \geq 0, m = n^2\}$ is not a CFL.

*Proof.* Suppose, to get a contradiction, that $L$ is context-free. Let $s = a^p b^{p^2}$, where $p$ is the pumping length of $L$. Let $s = uvxyz$ be the division of $s$ specified by the Pumping Lemma. If either $v$ or $y$ contains both $a$'s and $b$'s, then pumping up will break the format of $a^*b^*$. If both $v$ and $y$ consist of only $a$'s, then pumping down breaks the condition $m \geq n^2$ (the condition $m = n^2$ can be split into the two conditions $m \geq n^2$ and $m \leq n^2$). Similarly if $v$ and $y$ consist of only $b$'s. The only possibility left is that $v$ consists entirely of $a$'s and $y$ consists entirely of $b$'s. Then $N_a(v) \geq 1$ and $N_b(y) \leq p$, so pumping up $p$ times results in $N_a(uv^{p+1}xy^{p+1}z) \geq 2p$ and $N_b(uv^{p+1}xy^{p+1}z) \leq 2p^2 < 4p^2 = (2p)^2$, breaking the condition $m \geq n^2$. Hence, we have a contradiction of the Lemma, so $L$ cannot be a CFL. □

3.3. $L = \{x_1\# \ldots \#x_k : k \geq 1, \text{ each } x_i \in \{0,1\}^*, \text{ and for some } i, j,\ x_i = x_j^{\mathcal{R}}\}$ is a CFL.

*Proof.* One approach is to construct a PDA that uses nondeterminism to guess which two pieces to match against each other. A more concise proof, which we will present, is via a CFG. Observe that for any string $s = x_1\# \ldots \#x_k \in L$, if $i \neq j$ (where $x_i = x_j^{\mathcal{R}}$), then $s$ is of the form $x(wyw^{\mathcal{R}})z$, where

- $x \in (\{a,b\}^*\#)^*$
- $w \in \{a,b\}^*$
- $y \in (\#\{a,b\}^*)^*\#$
- $z \in (\#\{a,b\}^*)^*$.

If, on the other hand, $i = j$ (where $x_i = x_j^{\mathcal{R}}$), then $s$ is of the form $xvz$, where $x$ and $z$ are as above, and $v \in \{a,b\}^*$ such that $v = v^{\mathcal{R}}$. We know how to generate each of these types of substrings, so combining them gives us the following CFG for $L$.

$$\begin{aligned}
S &\to S_1|S_2 \\
S_1 &\to XY''Z \\
S_2 &\to XVZ \\
X &\to XT\#|\varepsilon \\
Z &\to Z\#T|\varepsilon \\
Y'' &\to aY''a|bY''b|Y'\# \\
Y' &\to Y'\#T|\varepsilon \\
V &\to aVa|bVb|a|b|\varepsilon \\
T &\to Ta|Tb|\varepsilon \\
U &\to Ua|Ub|U\#|\varepsilon
\end{aligned}$$

Here, $S_1$ and $S_2$ generate the two categories into which we split $L$. $X$, $Z$ and $V$ generate the corresponding lowercases described above. $T$ generates $\{a,b\}^*$ and $U$ generates $\{a,b,\#\}^*$. And lastly, $Y'$ generates our $y$ above, less the # suffix, and $Y''$ generates the substring $wyw^{\mathcal{R}}$. □

3.4. $L = \{b_i\#b_{i+1} : i \geq 1\}$ is not a CFL.

*Proof.* Suppose, to get a contradiction, that $L$ is context-free. Let $s = 1^p0^p\#1^p0^{p-1}1$, where $p$ is the pumping length of $L$. Let $s = uvxyz$ be the division of $s$ specified by the Pumping Lemma. First note that $vy$ cannot contain #, otherwise pumping down will result in no #'s in the string, not allowing it to be in $L$. Alternately, if $v$ and $y$ (ignoring $\varepsilon$'s) are both on the same side of the #, then pumping down will make the number represented on that side too small to satisfy $L$. The only possibly remaining is that $v$ is on the left and $y$ is on the right of the #, and more specifically, that $v$ consists of some positive number of 0's and $y$ consists of some positive number of 1's. Then pumping down gives us $1^p0^{p-|v|}\#1^{p-|y|}0^{p-1}1$. Given this LHS, the RHS would have to be $1^p0^{p-|v|-1}1$, but it clearly is not since $|v|, |y| > 0$. Hence, we have a contradiction of the Lemma, so $L$ cannot be a CFL. □

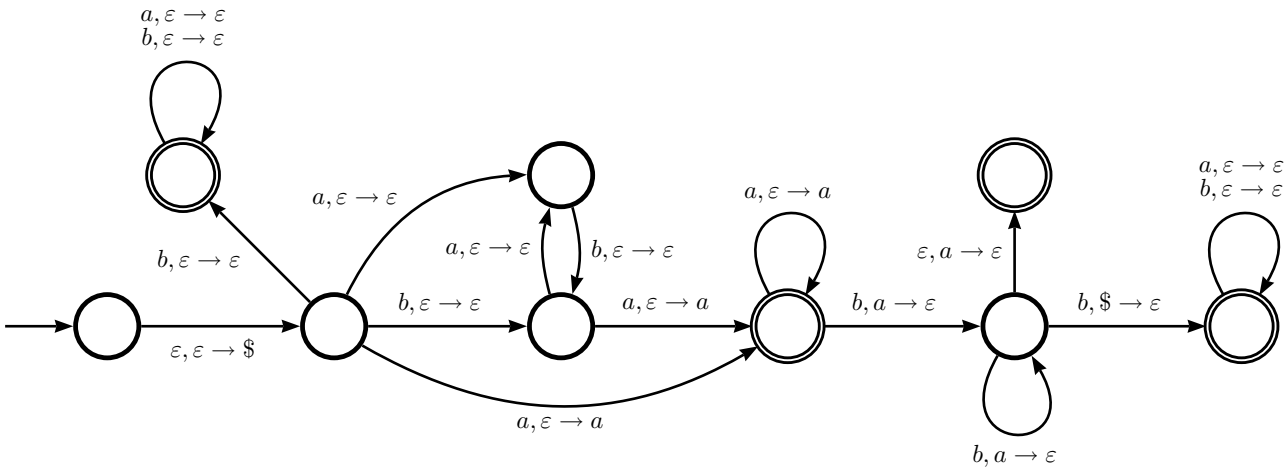
3.5. $L = (a \cup b)^* - \{(a^nb^n)^n : n \geq 1\}$ is a CFL.

*Proof.* We observe that any string in $\{a,b\}^*$ either is the empty string $\varepsilon$ or can be split into a positive number of non-empty "blocks" of the form $a^*b^*$, where all but the first and last blocks are in particular of the form $a^+b^+$. Now, a string $x \neq \varepsilon$ belongs to $L$ if and only if it satisfies at least one of the following conditions:

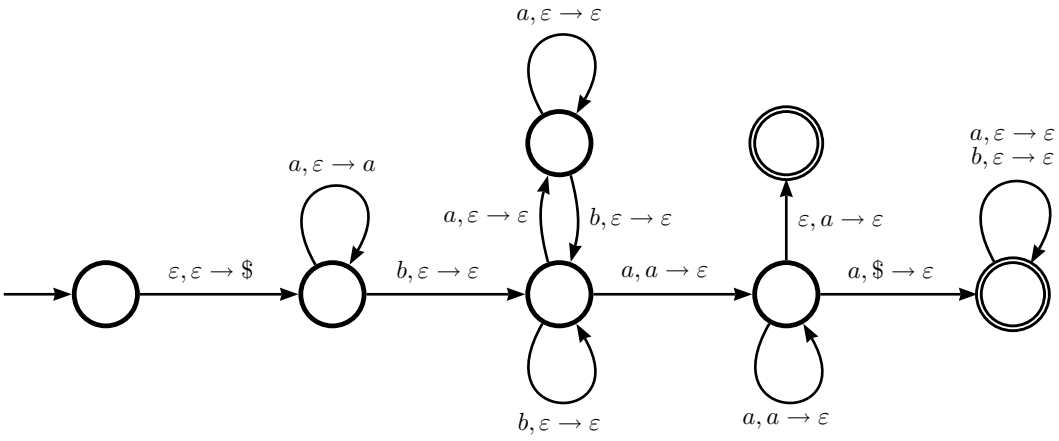
(1) some block has different numbers of $a$'s and $b$'s,
(2) some block has a different number of $a$'s from the first block, and
(3) the number of blocks is different from the number of $a$'s in the first block.

We can then write $L = L_1 \cup L_2 \cup L_3 \cup \{\varepsilon\}$, where $L_i$ consists of strings satisfying condition $(i)$. So by the closure of CFLs under union, it remains for us to show that $L_1$, $L_2$ and $L_3 \cup \{\varepsilon\}$ are CFLs.

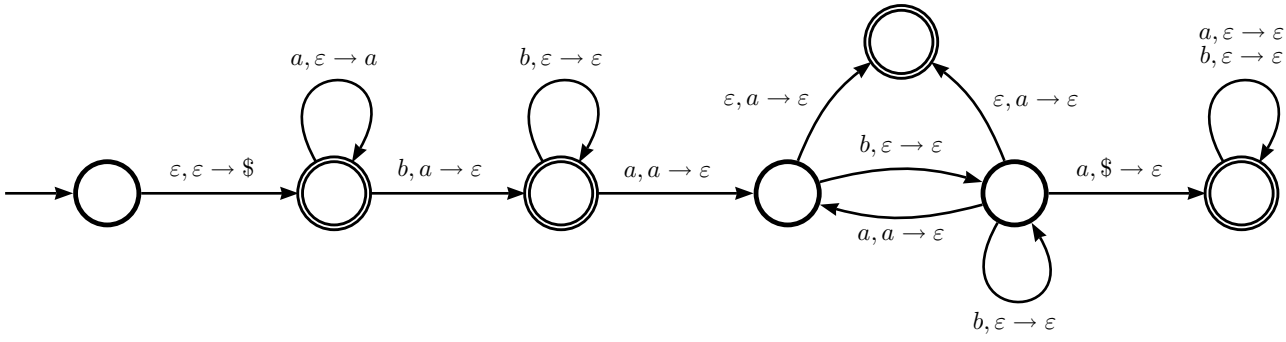
Here is a PDA for $L_1$:




Here is a PDA for $L_2$:




Here is a PDA for $L_3 \cup \{\varepsilon\}$:



It is left as an exercise for the reader to verify that these PDAs indeed recognize the languages specified. □

4. **(Complexity of CFGs)**

4.1. The complexity of $G$ is at most $O(|Q|^4)$.

*Proof.* First observe that every rule in $G$ has length at most $5 = O(1)$, so a bound on the number of rules is also a bound on the complexity of $G$. We consider each of the three types of rules separately.

Type I, of the form $A_{pq} \to aA_{rs}b$, consists of at most a rule for every choice of $p, q, r, s \in Q$, $t \in \Gamma$, and $a, b \in \Sigma_\varepsilon$, or a total of $|Q|^4 \cdot |\Gamma| \cdot (|\Sigma| + 1)^2 = O(|Q|^4)$ rules, since $|\Gamma|, |\Sigma| = O(1)$. Type II, of the form $A_{pq} \to A_{pr}A_{rq}$, consists of a rule for every choice of $p, q, r \in Q$, or a total of $|Q|^3$ rules. Type III, of the form $A_{pp} \to \varepsilon$, consists of a rule for every $p \in Q$, or a total of $|Q|$ rules. Tallying up all three, we get a total complexity of $O(|Q|^4)$.

Note that this bound is tight, since we can construct a PDA that has *every* allowed transition (possible thanks to non-determinism), which results in Type I containing $\Omega(|Q|^4)$ rules. □

4.2. *Proof.* Let $M = (Q, \Sigma, \delta, q_0, F)$ be our DFA, where $|Q| = n$. The idea is to have a variable for each state, which will generate all strings that take $M$ from that state to a final state. Our grammar is then $G = (Q, \Sigma, R, q_0)$, where $R$ is defined as follows.

- For each $q \in Q$, $a \in \Sigma$, include the rule $q \to a\delta(q, a)$.
- For each $q \in F$, include the rule $q \to \varepsilon$.

Each rule in $G$ is of $O(1)$ length, and there are a total of $|Q| \cdot |\Sigma| + |F| = O(n)$ rules, so the complexity of $G$ is $O(n)$. It remains to show that $G$ is indeed equivalent to $M$, that is, $\mathcal{L}(G) = \mathcal{L}(M)$. To do this, we prove the stronger claim that for each $q \in Q$, $w \in \Sigma^*$, $q \overset{*}{\Rightarrow} w$ if and only if $\hat{\delta}(q, w) \in F$, using induction on $|w|$.

If $|w| = 0$, then $w = \varepsilon$, so that $q \overset{*}{\Rightarrow} w$ if and only if $G$ contains the rule $q \Rightarrow \varepsilon$ if and only if $q \in F$ if and only if $\hat{\delta}(q, \varepsilon) = q \in F$. For our inductive case, suppose $|w| \geq 1$.

For our ($\Rightarrow$) direction, let $q \overset{*}{\Rightarrow} w$ and $a \in \Sigma$ be the first symbol of $w$. Then the first rule in the derivation must be $q \Rightarrow ap$, for some $p \in Q$. Now, we know $p \overset{*}{\Rightarrow} w[2 \ldots |w|]$, so by induction, $\hat{\delta}(p, w[2 \ldots |w|]) \in F$. But then $\hat{\delta}(q, w) = \hat{\delta}(p, w[2 \ldots |q|])$, so $\hat{\delta}(q, w) \in F$ as well.

For our ($\Leftarrow$) direction, suppose $\hat{\delta}(q, w) \in F$, and as above let $a \in \Sigma$ be the first symbol in $w$. Let $p = \delta(q, a)$, so that $\hat{\delta}(p, w[2 \ldots |w|]) \in F$. By induction, we know $p \overset{*}{\Rightarrow} w[2 \ldots |w|]$. But we also know $G$ contains the rule $q \to ap$, so we can combine these to get the derivation $q \Rightarrow ap \overset{*}{\Rightarrow} w$, completing our induction. □

5. **(Ambiguous grammars)**

5.1. Consider the following two distinct derivations for the same string:

```
< STMT > ⇒ <IF-THEN>
         ⇒ if condition then <STMT>
         ⇒ if condition then <IF-THEN-ELSE>
         ⇒ if condition then if condition then <STMT> else <STMT>
         ⇒ if condition then if condition then <ASSIGN> else <ASSIGN>
         ⇒ if condition then if condition then a:=1 else a:=1
```

$$
\begin{aligned}
\text{<STMT>} &\Rightarrow \texttt{<IF-THEN-ELSE>} \\
&\Rightarrow \texttt{if condition then <STMT> else <STMT>} \\
&\Rightarrow \texttt{if condition then <IF-THEN> else <STMT>} \\
&\Rightarrow \texttt{if condition then if condition then <STMT> else <STMT>} \\
&\Rightarrow \texttt{if condition then if condition then <ASSIGN> else <ASSIGN>} \\
&\Rightarrow \texttt{if condition then if condition then a:=1 else a:=1}
\end{aligned}
$$

5.2. Intuitively, this grammar is ambiguous because each "`else`" could be paired with any occurrence of "`if condition then`" that precedes it, and the string accordingly generated. Loosely speaking, think of this as parenthesis matching where every '`)`' has to be matched with a preceding '`(`', but some '`(`'s stand by itself. There are many ways to make such pairings, leading to ambiguity. To fix this, we need a convention to make the pairing unique and well-defined. We choose the following natural rule: from left to right, we pair each "`else`" with the closest preceding "`if condition then`" not already paired. We see then that between a paired "`if condition then`" and "`else`" there must be a completely matched sequence of '`(`'s and '`)`'s, that is, no stand-alone '`(`': if there had been an extra '`(`', it would have been used in a match before the '`(`' to its left.

We can now define the following unambiguous grammar. We introduce a new variable, `<MATCHED>`, corresponding to strings generated by the old `<STMT>` that are completely matched.

$$
\begin{aligned}
\texttt{<STMT>} &\rightarrow \texttt{<ASSIGN> | <IF-THEN> | <IF-THEN-ELSE>} \\
\texttt{<IF-THEN>} &\rightarrow \texttt{if condition then <STMT>} \\
\texttt{<IF-THEN-ELSE>} &\rightarrow \texttt{if condition then <MATCHED> else <STMT>} \\
\texttt{<MATCHED>} &\rightarrow \texttt{<ASSIGN> | if condition then <MATCHED> else <MATCHED>} \\
\texttt{<ASSIGN>} &\rightarrow \texttt{a:=1}
\end{aligned}
$$